WEB端PHP编码（安全）规范

本篇规范制定了代码基本元素的相关标准， 以确保共享的PHP代码间具有较高程度的技术互通性。

关键词 “必须”("MUST")、“一定不可/一定不能”("MUST NOT")、“需要”("REQUIRED")、 “将会”("SHALL")、“不会”("SHALL NOT")、“应该”("SHOULD")、“不该”("SHOULD NOT")、 “推荐”("RECOMMENDED")、“可以”("MAY")和”可选“("OPTIONAL")的详细描述可参见 RFC 2119 。

本文档主要基于PHP FIG(框架协同工作组) 提议的PSR:Proposing a Standards Recommendation(PHP编码规范) 整理修改，本规范主要参考psr2代码风格规范， 对命名空间不做要求，使用命名空间可能会导致无法支持国内一些框架，对日志接口规范不做要求。 可以用PHP_CodeSniffer 检查代码是否符合规范
示例：

```
phpcs --standard=PSR2 -n /path/to/php_source.php
```

1. 概览

- PHP代码**应该**与HTML代码分享，**推荐**使用MVC框架或模板引擎（smarty等等）
- PHP代码文件**必须**以 `<?php` 或 `<?=` 标签开始；
- PHP代码文件**必须**以 `不带BOM的 UTF-8` 编码；
- 所有PHP文件**必须**使用 `Unix LF (linefeed)` 作为行的结束符。
- 所有PHP文件**必须**以一个空白行作为结束。
- 纯PHP代码文件**必须**省略最后的 `?>` 结束标签。
- 代码**必须**使用4个空格符而不是 tab键 进行缩进。
- 每行的字符数**应该**软性保持在80个之内， 理论上**一定不可**多于120个， 但**一定不能**有硬性限制。
- **一定不能**用拼音命名
- 常量所有字母都**必须**大写，单词间用下划线分隔；
- PHP代码中**应该**只定义类、函数、常量等声明，或其他会产生 `从属效应` 的操作（如：生成文件输出以及修改.ini配置文件等），二者只能选其一；
- 类的命名**必须**遵循 `StudlyCaps` 大写开头的驼峰命名规范；
- 类中的常量所有字母都**必须**大写，单词间用下划线分隔；
- 方法名称**必须**符合 `camelCase` 式的小写开头驼峰命名规范。
- 类的属性和方法**必须**添加访问修饰符（ `private` 、 `protected` 以及 `public` ）， `abstract` 以及 `final` **必须**声明在访问修饰符之前，而 `static` **必须**声明在访问修饰符之后。
- 普通函数命名**可以**符合 `camelCase` 或 `under_score_case`
- 类的开始花括号(`{`)**必须**写在函数声明后自成一行，结束花括号(`}`)也**必须**写在函数主体后自成一行。
- 方法的开始花括号(`{`)**必须**写在函数声明后自成一行，结束花括号(`}`)也**必须**写在函数主体后自成一行。
- 控制结构的关键字后**必须**要有一个空格符，而调用方法或函数时则**一定不能**有。
- 控制结构的开始花括号(`{`)**必须**写在声明的同一行，而结束花括号(`}`)**必须**写在主体后自成一行。
- 控制结构的开始左括号后和结束右括号前，都**一定不能**有空格符。

1.1. 例子

以下例子程序简单地展示了以上大部分规范：

```php
<?php
/**
 * start page for standard coding
 *
 * PHP version 5
 *
 * @category  PHP
 * @package   Standard_Coding
 * @author    Seven Heart <stvenx@gmail.com>
 * @copyright 2014 UNNOO
 * @license   http://opensource.org/licenses/gpl-2.0.php GNU General Public License
 * @version   v0.31
 * @link      http://www.unnoo.com
 */
namespace Vendor\Package;

use BarClass as Bar;
use FooInterface;
use OtherVendor\OtherPackage\BazClass;

/**
 * implements FooInterface
 *
 * @category  PHP
 * @package   Standard_Coding
 * @author    Seven Heart <stvenx@gmail.com>
 * @copyright 2014 UNNOO
 * @license   http://opensource.org/licenses/gpl-2.0.php GNU General Public License
 * @version   v0.31
 * @link      http://www.unnoo.com
 */
class Foo extends Bar implements FooInterface
{
    /**
     * sampleFunction 函数描述
     *
     * @param string $a 参数$a描述
     * @param string $b 参数$b描述
     *
     * @return type    返回值
     */
    public function sampleFunction($a, $b = null)
    {
        $varName = null;
        if ($a === $b) {
            bar();
        } elseif ($a > $b) {
            $foo->bar($arg1);
        } else {
            BazClass::bar($arg2, $arg3);
        }
    }

    /**
     * example function bar
     *
     * @param string $str 传入字符串
     *
     * @return void
     */
    final public static function bar($str)
    {
        // method body
    }
}
```

2. 文件

2.1. PHP标签

PHP代码**必须**使用 `<?php ?>` 长标签 或 `<?= ?>` 短输出标签； **一定不可**使用其它自定义标签。

2.2. 字符编码

PHP代码**必须**且只可使用`不带BOM的UTF-8`编码。

2.3. 从属效应（副作用）

一份PHP文件中**应该**要不就只定义新的声明，如类、函数或常量等不产生从属效应的操作，要不就只有会产生从属效应的逻辑操作，但**不该**同时具有两者。

“从属效应”(side effects)一词的意思是，仅仅通过包含文件，不直接声明类、 函数和常量等，而执行的逻辑操作。

“从属效应”包含却不仅限于：生成输出、直接的 `require` 或 `include` 、连接外部服务、修改 ini 配置、抛出错误或异常、修改全局或静态变量、读或写文件等。

以下是一个反例，一份包含声明以及产生从属效应的代码：

```php
<?php
// 从属效应：修改 ini 配置
ini_set('error_reporting', E_ALL);

// 从属效应：引入文件
include "file.php";

// 从属效应：生成输出
echo "<html>\n";

// 声明函数
function foo()
{
    // 函数主体部分
}
```

下面是一个范例，一份只包含声明不产生从属效应的代码：

```php
<?php
// 声明函数
function foo()
{
    // 函数主体部分
}

// 条件声明**不**属于从属效应
if (! function_exists('bar')) {
    function bar()
    {
        // 函数主体部分
    }
}
```

4. 通则

4.1. 行

行的长度**一定不能**有硬性的约束。

软性的长度约束**一定**要限制在120个字符以内，若超过此长度，带代码规范检查的编辑器**一定**要发出警告，不过**一定不可**发出错误提示。

每行**不应该**多于80个字符，大于80字符的行**应该**折成多行。

非空行后**一定不能**有多余的空格符。

空行**可以**使得阅读代码更加方便以及有助于代码的分块。

每行**一定不能**存在多于一条语句。

4.2. 缩进

代码**必须**使用4个空格符的缩进，**一定不能**用 tab键 。

> 备注: 使用空格而不是tab键缩进的好处在于， 避免在比较代码差异、打补丁、重阅代码以及注释时产生混淆。 并且，使用空格缩进，让对齐变得更方便。

4.3. 关键字 以及 True/False/Null

PHP所有 关键字**必须**全部小写。

常量 `true` 、 `false` 和 `null` 也**必须**全部小写。

5. namespace 以及 use 声明

`namespace` 声明后 必须 插入一个空白行。

所有 `use` 必须 在 `namespace` 后声明。

每条 `use` 声明语句 必须 只有一个 `use` 关键词。

`use` 声明语句块后 必须 要有一个空白行。

例如:

```php
<?php
namespace Vendor\Package;

use FooClass;
use BarClass as Bar;
use OtherVendor\OtherPackage\BazClass;

// ... additional PHP code ...
```

6. 类、类的常量、属性和方法

此处的“类”指代所有的类、接口以及可复用代码块（traits）

6.1. 常量

类的常量中所有字母都**必须**大写，词间以下划线分隔。 参照以下代码:

```php
<?php

class Foo
{
    const VERSION = '1.0';
    const DATE_APPROVED = '2012-06-01';
}
```

4.1. 扩展与继承

关键词 `extends` 和 `implements` **必须**写在类名称的同一行。

类的开始花括号**必须**独占一行，结束花括号也**必须**在类主体后独占一行。

```php
<?php
```

```php
namespace Vendor\Package;

use FooClass;
use BarClass as Bar;
use OtherVendor\OtherPackage\BazClass;

class ClassName extends ParentClass implements \ArrayAccess, \Countable
{
    // constants, properties, methods
}
```

`implements` 的继承列表也**可以**分成多行，这样的话，每个继承接口名称都**必须**分开独立成行，包括第一个。

```php
<?php
namespace Vendor\Package;

use FooClass;
use BarClass as Bar;
use OtherVendor\OtherPackage\BazClass;

class ClassName extends ParentClass implements
    \ArrayAccess,
    \Countable,
    \Serializable
{
    // constants, properties, methods
}
```

6.2. 属性

类的属性命名可以遵循 大写开头的驼峰式 (`$StudlyCaps`)、小写开头的驼峰式 (`$camelCase`) 又或者是 下划线分隔式 (`$under_score`)，本规范不做强制要求，但无论遵循哪种命名方式，都**应该**在一定的范围内保持一致。这个范围可以是整个团队、整个包、整个类或整个方法。

每个属性都**必须**添加访问修饰符。

一定不可使用关键字 `var` 声明一个属性。

每条语句**一定不可**定义超过一个属性。

不要使用下划线作为前缀，来区分属性是 protected 或 private。

以下是属性声明的一个范例：

```php
<?php
namespace Vendor\Package;

class ClassName
{
    public $foo = null;
}
```

6.3. 方法

方法名称**必须**符合 `camelCase()` 式的小写开头驼峰命名规范。 此处的“类”泛指所有的class类、接口以及traits可复用代码块。

所有方法都**必须**添加访问修饰符。

不要使用下划线作为前缀，来区分方法是 protected 或 private。

方法名称后**一定不能**有空格符，其开始花括号**必须**独占一行，结束花括号也**必须**在方法主体后单独成一行。参数左括号后和右括号前**一定不能**有空格。

一个标准的方法声明可参照以下范例，留意其括号、逗号、空格以及花括号的位置。

```php
<?php
namespace Vendor\Package;
```

```php
class ClassName
{
    public function fooBarBaz($arg1, &$arg2, $arg3 = [])
    {
        // method body
    }
}
```

6.4. 方法的参数

参数列表中，每个参数后面**必须**要有一个空格，而前面**一定不能**有空格。

有默认值的参数，**必须**放到参数列表的末尾。

```php
<?php
namespace Vendor\Package;

class ClassName
{
    public function foo($arg1, &$arg2, $arg3 = [])
    {
        // method body
    }
}
```

参数列表**可以**分列成多行，这样，包括第一个参数在内的每个参数都**必须**单独成行。

拆分成多行的参数列表后，结束括号以及方法开始花括号 必须 写在同一行，中间用一个空格分隔。

```php
<?php
namespace Vendor\Package;

class ClassName
{
    public function aVeryLongMethodName(
        ClassTypeHint $arg1,
        &$arg2,
        array $arg3 = []
    ) {
        // method body
    }
}
```

6.5. `abstract` 、 `final` 、以及 `static`

需要添加 `abstract` 或 `final` 声明时， **必须**写在访问修饰符前，而 `static` 则**必须**写在其后。

```php
<?php
namespace Vendor\Package;

abstract class ClassName
{
    protected static $foo;

    abstract protected function zim();

    final public static function bar()
    {
        // method body
    }
}
```

6.6. 方法及函数调用

方法及函数调用时，方法名或函数名与参数左括号之间**一定不能**有空格，参数右括号前也 **一定不能**有空格。每个参数前**一定不能**有空格，但其后**必须**有一个空格。

```php
<?php
bar();
$foo->bar($arg1);
Foo::bar($arg2, $arg3);
```

参数**可以**分列成多行，此时包括第一个参数在内的每个参数都**必须**单独成行。

```php
<?php
$foo->bar(
    $longArgument,
    $longerArgument,
    $muchLongerArgument
);
```

7. 控制结构

控制结构的基本规范如下：

- 控制结构关键词后**必须**有一个空格。
- 左括号 `(` 后**一定不能**有空格。
- 右括号 `)` 前也**一定不**能有空格。
- 右括号 `)` 与开始花括号 `{` 间**一定**有一个空格。
- 结构体主体**一定**要有一次缩进。
- 结束花括号 `}` **一定**在结构体主体后单独成行。

每个结构体的主体都**必须**被包含在成对的花括号之中， 这能让结构体更加结构话，以及减少加入新行时，出错的可能性。

7.1. `if` 、 `elseif` 和 `else`

标准的 `if` 结构如下代码所示，留意 括号、空格以及花括号的位置， 注意 `else` 和 `elseif` 都与前面的结束花括号在同一行。

```php
<?php
if ($expr1) {
    // if body
} elseif ($expr2) {
    // elseif body
} else {
    // else body;
}
```

应该使用关键词 `elseif` 代替所有 `else if` ，以使得所有的控制关键字都像是单独的一个词。

7.2. `switch` 和 `case`

标准的 `switch` 结构如下代码所示，留意括号、空格以及花括号的位置。 `case` 语句**必须**相对 `switch` 进行一次缩进，而 `break` 语句以及 `case` 内的其它语句都 必须 相对 `case` 进行一次缩进。 如果存在非空的 `case` 直穿语句，主体里必须有类似 `// no break` 的注释。

```php
<?php
switch ($expr) {
    case 0:
        echo 'First case, with a break';
        break;
    case 1:
        echo 'Second case, which falls through';
        // no break
    case 2:
    case 3:
    case 4:
        echo 'Third case, return instead of break';
```

```
        return;
    default:
        echo 'Default case';
        break;
}
```

7.3. `while` 和 `do while`

一个规范的 `while` 语句应该如下所示，注意其 括号、空格以及花括号的位置。

```php
<?php
while ($expr) {
    // structure body
}
```

标准的 `do while` 语句如下所示，同样的，注意其 括号、空格以及花括号的位置。

```php
<?php
do {
    // structure body;
} while ($expr);
```

7.4. `for`

标准的 `for` 语句如下所示，注意其 括号、空格以及花括号的位置。

```php
<?php
for ($i = 0; $i < 10; $i++) {
    // for body
}
```

7.5. `foreach`

标准的 `foreach` 语句如下所示，注意其 括号、空格以及花括号的位置。

```php
<?php
foreach ($iterable as $key => $value) {
    // foreach body
}
```

7.6. `try`, `catch`

标准的 `try catch` 语句如下所示，注意其 括号、空格以及花括号的位置。

```php
<?php
try {
    // try body
} catch (FirstExceptionType $e) {
    // catch body
} catch (OtherExceptionType $e) {
    // catch body
}
```

8. 闭包

闭包声明时，关键词 `function` 后以及关键词 `use` 的前后都**必须**要有一个空格。

开始花括号**必须**写在声明的同一行，结束花括号**必须**紧跟主体结束的下一行。

参数列表和变量列表的左括号后以及右括号前，**必须不能**有空格。

参数和变量列表中，逗号前**必须不能**有空格，而逗号后**必须**要有空格。

闭包中有默认值的参数**必须**放到列表的后面。

标准的闭包声明语句如下所示，注意其 括号、逗号、空格以及花括号的位置。

```php
<?php
$closureWithArgs = function ($arg1, $arg2) {
    // body
};

$closureWithArgsAndVars = function ($arg1, $arg2) use ($var1, $var2) {
    // body
};
```

参数列表以及变量列表**可以**分成多行，这样，包括第一个在内的每个参数或变量都**必须**单独成行，而列表的右括号与闭包的开始花括号**必须**放在同一行。

以下几个例子，包含了参数和变量列表被分成多行的多情况。

```php
<?php
$longArgs_noVars = function (
    $longArgument,
    $longerArgument,
    $muchLongerArgument
) {
   // body
};

$noArgs_longVars = function () use (
    $longVar1,
    $longerVar2,
    $muchLongerVar3
) {
   // body
};

$longArgs_longVars = function (
    $longArgument,
    $longerArgument,
    $muchLongerArgument
) use (
    $longVar1,
    $longerVar2,
    $muchLongerVar3
) {
   // body
};

$longArgs_shortVars = function (
    $longArgument,
    $longerArgument,
    $muchLongerArgument
) use ($var1) {
   // body
};

$shortArgs_longVars = function ($arg) use (
    $longVar1,
    $longerVar2,
    $muchLongerVar3
) {
   // body
};
```

注意，闭包被直接用作函数或方法调用的参数时，以上规则仍然适用。

```php
<?php
$foo->bar(
```

```
    $arg1,
    function ($arg2) use ($var1) {
        // body
    },
    $arg3
);
```

9.注释文档

所有文档块 ("docblocks") 必须和 phpDocumentor 格式兼容，phpDocumentor 格式的描述超出了本文档的范围，关于它的详情，参考：» http://phpdoc.org/。

所有类文件必须在文件的顶部包含文件级 （"file-level"）的 docblock ， 在每个类的顶部放置一个 "class-level" 的 docblock。下面是一些例子：

9.1. 文件

每个包含 PHP 代码的文件必须至少在文件顶部的 docblock 包含这些 phpDocumentor 标签：

```
/**
 * 文件的简短描述
 *
 * 文件的详细描述（如果有的话）... ...
 *
 * LICENSE: 一些 license 信息
 *
 * @copyright  Copyright (c) 2005-2014 Zend Technologies USA Inc. (http://www.zend.com)
 * @license    http://framework.zend.com/license/3_0.txt   BSD License
 * @version    $Id:$
 * @link       http://framework.zend.com/package/PackageName
 * @since      File available since Release 1.5.0
*/
```

9.2. 类

每个类必须至少包含这些 phpDocumentor 标签：

```
/**
 * 类的简述
 *
 * 类的详细描述 （如果有的话）... ...
 *
 * @copyright  Copyright (c) 2005-2014 Zend Technologies USA Inc. (http://www.zend.com)
 * @license    http://framework.zend.com/license/   BSD License
 * @version    Release: @package_version@
 * @link       http://framework.zend.com/package/PackageName
 * @since      Class available since Release 1.5.0
 * @deprecated Class deprecated in Release 2.0.0
 */
```

9.3. 函数

每个函数，包括对象方法，必须有最少包含下列内容的文档块（docblock）：

函数的描述

所有参数

所有可能的返回值

```
/**
 * @param boolean $hi when true 'Hello world' is echo-ed.
 *
 * @return void
 */
```

```
function outputHello($quiet)
{
    if ($quiet} {
        return;
    }
    echo 'Hello world';
}
```

因为访问级已经通过 "public"、 "private" 或 "protected" 声明， 不需要使用 "@access"。

10. 安全编码规范

原则，不信任所有从客户端传入的数据，包括 $_SERVER, $_GET, $_POST, $_COOKIE, $_FILES, $_ENV, $_REQUEST

10.1. SQL注入

目前最有佳的防止SQL注入的方法是使用SQL预编译， php提供的PDO/MySQLi可使用SQL预编译功能， **推荐** php_mysqli扩展，一个MySQLi封装类：MysqliDb

一定不能直接拼接用户提交的数据查询

问题代码示例：

```
<?php
$id = $_GET['id'];
$conn = mysql_connect("localhost", "root", "") or die("wrong!");
$sel = mysql_select_db("mydb", $conn);
$sql = "select * from user where id = " . $id;
$result = mysql_query($sql, $conn);
```

安全代码示例：

```
<?php
$id = $_GET['id'];
$conn = mysql_connect("localhost", "root", "") or die("wrong!");
$sel = mysql_select_db("mydb", $conn);
$sql = "select * from user where id = :id";
$stmt = $conn->prepare($sql);
$stmt->execute(array(':id' => $id));
```

10.2. XSS（跨站脚本攻击）

对参数做html转义过滤（要过滤的字符包括：单引号、双引号、大于号、小于号，&符号），防止脚本执行。在变量输出时进行HTML ENCODE处理。 可以使用htmlspecialchars 或filter_var_array对用户参数进行编码； 以下代码可以防止一般的xss

```
<?php
$input = filter_var_array($_POST, FILTER_SANITIZE_SPECIAL_CHARS);
```

10.3. CSRF（伪造跨站请求）

使用csrf token的验证机制来保证用户请求的合法性：即在功能请求页面设置csrf token，并在服务端对token进行验证。

10.4. 文件上传

强制检查文件后缀，保存文件时按安全规则（随机字符串等等）强制重命名，防止文件按非预期文件名保存
上传目录应该**一定不可**有执行权限
如果有条件**应该**单独设置文件域名

10.5. 运维安全

- 所使用组件没有已知漏洞；
- 第三方组件不允许使用默认文件名或目录名称，必须改名后使用；
- 与程序功能无关的文件不允许上传至生产服务器，例如包含服务器账号信息的readme.txt或help.txt等；
- 用于测试或调试的工具、脚本不允许上传至生产服务器；
- 所有文件必须确保在通过浏览器直接访问的情况下，不造成意外后果，同时也不返回任何有利于攻击者做进一步尝试的内容；
- 用到的第三方组件向安全运维人员备案，以便统一监测安全公告。